Purpose: The purpose of this assignment is to further your understanding of how a hash table works.

Instructions:

- For Part 1, complete in a Word document.
- Complete Part 2, work in a copy of your code from class.
- For your submission, submit a single zip file containing the Word document for Part 1, and all the C# files you changed for your work in Part 2. Both the Word document and all the C# files should be in the root of the zip file, no subfolders.
- You may work in a pair for this assignment, if you so choose. If you do so, only one of the pair should submit but ensure it is prom
- Due Date: Tuesday, November 18th, at 10:00pm.

Part I – Hash Table Traces [20 Marks]

Part I requires you to trace through the processing of a hash table. Show what the hashtable would look like at each step as you add items.

For this question, you will be storing strings in a hash table. The hash function simply adds up the position of each character of the name as it appears in the alphabet and is not case sensitive. Hash tables expand when they **exceed** the load factor given.

A	B	C	D	…
1	2	3	4	…

The following strings will be added, in the order they appear, to the hash table:
SALLY, ROB, RICK, BRYCE, JOAN, BILL

For both parts below, show each intermediate tables as it would appear just before expanding.

a. Add the names to a table that utilizes chaining (with linked lists for collisions). The initial table size is 4 and the table doubles in size when the load factor of 72% is reached.
b. Add the names to a table that utilizes open-addressing with the following implementations. The table sizes are: 5, 7, 11, 31 (may not need all sizes).
    i. Linear – Load Factor = 0.72, Increment by 1.
    ii. Quadratic – Load Factor = 0.5, $C_1 = 1$ and $C_2 = 1$
    iii. Double Hash – Load Factor = 0.72

Part 2 – Programming [35 Marks]

1. Implement a Chaining implementation of a hash table. For this question, you will use the BST written in class as your secondary storage structure. Note that an AVLT could be used, but is not necessary as the trees used will be relatively small. [25 Marks]

Your new Chaining Hashtable class should be called ChainingBST.

You will implement the following methods:

a. Add,
b. Remove,
c. Get
d. GetEnumerator
e. ToString (a string representation of all the key-value pairs in the hashtable)

Hints:

- Make sure your BST code is working properly.
- You should obviously extend and fully implement A_Hashtable at the very least. I will expect that any tests written for an object of type A_Hashtable should pass on your new hashtable
    - (yes, that's a hint that you can use the test methods we developed in class to test your hashtable – and that I will likely do the same)
- Since chaining cannot get into an infinite loop when accessing the hash table, we don't need to have prime number table sizes. Simply start your hash table at 5 elements and double.

2. The Get method of our open addressing hashtable is coded to skip over tombstones while looking for an element in the hash table. It would be more efficient to move a found item "up" the collision chain to where the first tombstone is located in the collision chain. Modify the Get method to implement this functionality in an efficient manner. [10 Marks]

Once again, you should ensure that any hashtable unit tests will still all pass on your modified open addressing hashtable.